案例 19-直线 Liang-Barsky 裁剪算法

文档编写：霍波魏

校稿/修订：孔令德

时间 2019~2020

联系方式：QQ997796978

说明：本套案例由孔令德开发，原版本为 Visual C++6.0，配套于孔令德的著作《计算机图形学-基于 MFC 三维图形开发》一书。孔令德计算机工程研究所的学生霍波魏在学习计算机图形学期间，对本套案例进行了升级并编写了学习文档。现在程序的编写和程序的解释都是基于 Windows 10 操作系统，使用 Microsoft visual studio 2017 平台的 MFC（英文版）开发。

一、知识点

Liang-Barsky 裁剪算法原理：Liang-Barsky 裁剪算法把直线段与窗口的相互位置关系划分为两种情况进行讨论：平行与窗口边界的直线段与不平行于窗口边界的直线段。

设起点为$P_0(X_0,Y_0)$，终点为$P_1(X_1,Y_1)$的直线段参数方程为：

$$P = P_0 + t(P_1 - P_0)$$

展开形式为

$$X = X_0 + t(X_1 - X_0)$$

$$Y = Y_0 + t(Y_1 - Y_0)$$

式中 0≤t≤1。

二、案例描述

本案例使用直线 Liang-Barsky 裁剪算法，通过绘制放大镜窗口对金刚石团进行局部裁剪放大。

三、实现步骤

1. 在 CTestView 中添加绘制放大镜窗口函数、坐标系转换函数、绘制金刚石图案函数、裁剪函数、裁剪测试函数。

2. 在 CTestView 中添加消息函数，在 OnDraw 中调用 DoubleBuffer 函数定义客户区坐标、绘制裁剪窗口。

四、主要算法

CTestView 类

```
public:
    void DoubleBuffer(CDC* pDC);//双缓冲函数
    void DrawRect(CDC* pDC);//绘制放大镜窗口
    CP2i Convert(CPoint point);//坐标系转换
    void Diamond(CDC *pDC, BOOL bclip);//绘制金刚石图案函数
    BOOL LBLineClip();//裁剪函数
    BOOL ClipTest(double u, double v, double &tmax, double &tmin);
        //裁剪测试函数
protected:
    int nClientWidth;//屏幕客户区宽度
    int nClientHeight;//屏幕客户区高度
    int nHWidth, nHHeight;//屏幕客户区的半宽和半高
    CLine* line;//直线的指针
    CP2d  P[2];//直线的起点和终点
    CP2d* V;//动态定义等分点数组
    CP2i nRectCenter;//裁剪矩形的中心坐标
    int nRectHalfHeight, nRectHalfWidth;//裁剪矩形的半高度和半宽度
    int Wxl, Wxr, Wyt, Wyb;//窗口左上角点与右下角点坐标
void CTestView::DoubleBuffer(CDC* pDC)//双缓冲函数
{
    CRect rect;//定义客户区
    GetClientRect(&rect);//获得客户区的大小
    nClientWidth = rect.Width();//屏幕客户区宽度
    nClientHeight = rect.Height();//屏幕客户区高度
    nHWidth = nClientWidth / 2;//屏幕客户区半宽
    nHHeight = nClientHeight / 2;//屏幕客户区半高
    CDC memDC;
    memDC.CreateCompatibleDC(pDC);
    CBitmap NewBitmap, *pOldBitmap;
    NewBitmap.CreateCompatibleBitmap(pDC, nClientWidth, nClientHeight);
    pOldBitmap = memDC.SelectObject(&NewBitmap);
    memDC.FillSolidRect(&rect, pDC->GetBkColor());
    Diamond(&memDC, FALSE);//绘制金刚石背景图
    DrawRect(&memDC);//绘制放大镜
    Diamond(&memDC, TRUE);//绘制裁剪窗口中的金刚石
    pDC->BitBlt(0, 0, nClientWidth, nClientHeight, &memDC, 0, 0, SRCCOPY);
                                    //将内存位图拷贝到屏幕
    memDC.SelectObject(pOldBitmap);//恢复位图
    NewBitmap.DeleteObject();//删除位图
}
void CTestView::DrawRect(CDC* pDC)//绘制放大镜窗口
{
    CRGB LineClr = CRGB(0.0, 0.5, 0.0);
```

```
    line->MoveTo(pDC, nHWidth + Wxl, nHHeight - Wyt, LineClr);
    line->LineTo(pDC, nHWidth + Wxr, nHHeight - Wyt, LineClr, 3);
    line->LineTo(pDC, nHWidth + Wxr, nHHeight - Wyb, LineClr, 3);
    line->LineTo(pDC, nHWidth + Wxl, nHHeight - Wyb, LineClr, 3);
    line->LineTo(pDC, nHWidth + Wxl, nHHeight - Wyt, LineClr, 3);
}
CP2i CTestView::Convert(CPoint point)//坐标系转换
{
    CP2i ptemp;
    ptemp.x = point.x - nHWidth;
    ptemp.y = nHHeight - point.y;
    return ptemp;
}
void CTestView::Diamond(CDC *pDC, BOOL bclip)//绘制金刚石图案函数
{
    double Theta;//Theta为圆的等分角
    int n = 18;//定义圆的等分点
    V = new CP2d[n];
    double r = 300;//定义圆的半径
    Theta = 2 * PI / n;
    for (int i = 0; i < n; i++)//计算等分点坐标
    {
        V[i].x = r * cos(i*Theta);
        V[i].y = r * sin(i*Theta);
    }
    for (int i = 0; i <= n - 2; i++)//依次连接各等分点
    {
        for (int j = i + 1; j <= n - 1; j++)
        {
            if (!bclip)
            {
                line->MoveTo(pDC, ROUND(nHWidth + V[i].x),
                    ROUND(nHHeight - V[i].y));
                line->LineTo(pDC, ROUND(nHWidth + V[j].x),
                    ROUND(nHHeight - V[j].y));
            }
            else
            {
                P[0] = V[i];
                P[1] = V[j];
                if (LBLineClip())
                {
                    line->MoveTo(pDC, ROUND(nHWidth + P[0].x),
                        ROUND(nHHeight - P[0].y));
```

```
                    line->LineTo(pDC, ROUND(nHWidth + P[1].x),
                        ROUND(nHHeight - P[1].y), 3);
                }
            }
        }
    }
    delete[]V;
}
BOOL CTestView::LBLineClip()//裁剪函数
{
    double tmax, tmin, dx, dy;
    dx = P[1].x - P[0].x;
    dy = P[1].y - P[0].y;
    tmax = 0.0, tmin = 1.0;
    //窗口边界的左、右、下、上顺序裁剪直线
    if (ClipTest(-dx, P[0].x - Wxl, tmax, tmin))
            //n=1,左边界u1=-△x，v1=x0-Wxl
    {
        if (ClipTest(dx, Wxr - P[0].x, tmax, tmin))
                    //n=2，右边界u2=△x，v2=Wxr-x0
        {
            if (ClipTest(-dy, P[0].y - Wyb, tmax, tmin))
                    //n=3，下边界u3=-△y，v3=y0-Wyb
            {
                if (ClipTest(dy, Wyt - P[0].y, tmax, tmin))
                    //n=4，上边界u4=△y，v4=Wyt-y0
                {
                    if (tmin < 1.0)//判断直线的终点
                    {
                        P[1].x = P[0].x + tmin * dx;//重新计算直线终点坐标
                        P[1].y = P[0].y + tmin * dy;//x=x0+t(x1-x0)格式
                    }
                    if (tmax > 0.0)//判断直线的起点
                    {
                        P[0].x = P[0].x + tmax * dx;//重新计算直线起点坐标
                        P[0].y = P[0].y + tmax * dy;//x=x0+t(x1-x0)格式
                    }
                    return TRUE;
                }
            }
        }
    }
    return FALSE;
}
```

```
BOOL CTestView::ClipTest(double u, double v, double &tmax, double &tmin)//裁剪测试函数
{
    double t;
    BOOL ReturnValue = TRUE;
    if (u < 0.0)//从裁剪窗口的外部到内部,计算起点处的tmax
    {
        t = v / u;
        if (t > tmin)
            ReturnValue = FALSE;
        else if (t > tmax)
            tmax = t;
    }
    else
    {
        if (u > 0.0)//从裁剪窗口的内部到外部，计算终点处的tmin
        {
            t = v / u;
            if (t < tmax)
                ReturnValue = FALSE;
            else if (t < tmin)
                tmin = t;
        }
        else//平行于窗口边界的直线
            if (v < 0.0)//直线在窗口外可直接删除
                ReturnValue = FALSE;
    }
    return(ReturnValue);
}
void CTestView::OnMouseMove(UINT nFlags, CPoint point)
{
    // TODO: 在此添加消息处理程序代码和/或调用默认值
    nRectCenter = Convert(point);//鼠标指针为放大镜中心
    Wxl = nRectCenter.x - nRectHalfWidth;//窗口的左边界
    Wxr = nRectCenter.x + nRectHalfWidth;//窗口的右边界
    Wyb = nRectCenter.y - nRectHalfHeight;//窗口的下边界
    Wyt = nRectCenter.y + nRectHalfHeight;//窗口的上边界
    Invalidate(FALSE);
    CView::OnMouseMove(nFlags, point);
}
void CTestView::OnDrawpic()
{
    // TODO: 在此添加命令处理程序代码
    MessageBox(_T("请移动鼠标观察图形！"), _T("提示"), MB_ICONEXCLAMATION |
```

```
MB_OK);
}
BOOL CTestView::OnEraseBkgnd(CDC* pDC)
{
    // TODO: 在此添加消息处理程序代码和/或调用默认值
    return TRUE;
}
```

五、实现效果

直线 Liang-Barsky 裁剪算法效果如图 19-1 和 19-2 所示。

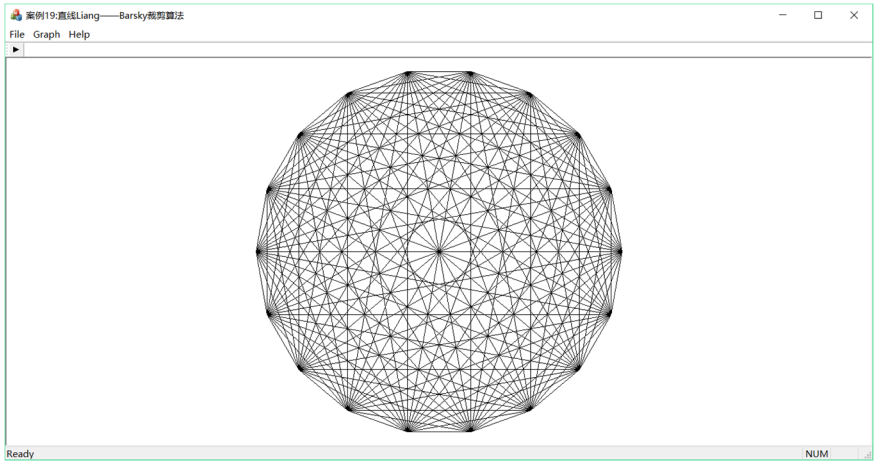

图 19-1 直线 Liang-Barsky 裁剪算法效果图一

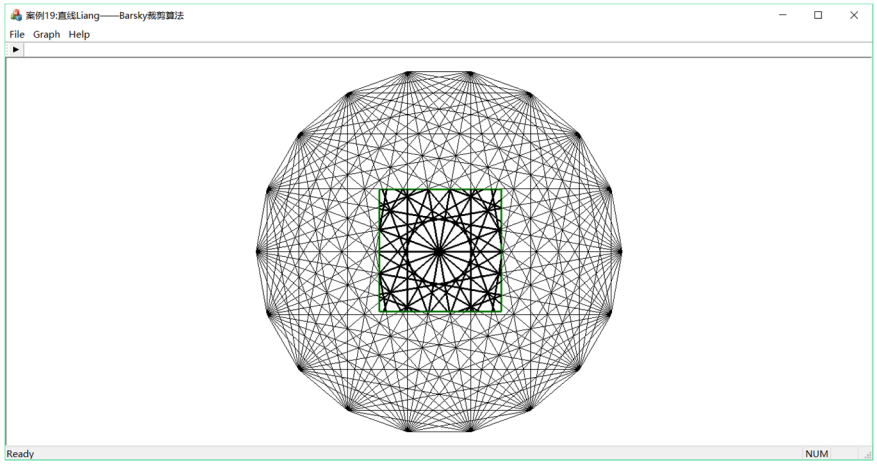

图 19-2 直线 Liang-Barsky 裁剪算法效果图二

六、补充

Liang-Barsky 算法是比 Cohen-Sutherland 算法更快的直线裁剪算法，他将二维裁剪问题化成两次一维裁剪问题，而把裁剪问题转化为解一组不等式的问题；改善了 Cohen-Sutherland 算法中全部摒弃的判断只适用于那些仅在窗口同一侧线段的不足之处。Liang-Barsky 算法主要考虑直线段方程参数 t 的变化情况。裁剪窗口的每条边界线将平面分为两个区域。定义参见窗口所在侧为可见侧，另一侧

为不可见侧。$U_n < 0$表示直线段从裁剪窗口及其边界延长线的不可见侧延伸到可见侧，直线段的窗口边界的交点位于直线段的起点一侧；$U_n > 0$表示直线段从裁剪窗口及其边界延长线的可见侧延伸到不可见侧，直线段于窗口边界的交点位于直线段的终点一侧。对于直线段的起点一侧，如果当$U_n < 0$时，被裁剪直线段的起点取 t 的最大值t_{max}，对于直线段的终点一侧，如果当$U_n > 0$时，被裁减直线段的终点取 t 的最小值t_{min}；如果$t_{max} \leq t_{min}$，则被裁减的直线段位于窗口内；如果$t_{max} > t_{min}$，则被裁减的直线段位于窗口外。存在$U_n < 0$时，t_{max}应该大于 0；在$U_n > 0$时，t_{min}应该小于 1。即：

$$\begin{cases} t_{max} = \max(0, t_n | U_n < 0) \\ t_{min} = \min(t_n | U_n > 0, 1) \end{cases}$$